The purpose of this lab is to give you experience with authenticating users and infosec.

Tips

- An auto-generated ERD is on the last page with information about the database. Reference it as needed.

Specifications

1. Download the InfoSecDemo database from Canvas and restore it in a MySQL VM/container.
    a. If using a Docker container, open a new terminal/PowerShell instance. You can copy files to your container with the following command:

        `docker cp <file path and name> <container ID>:/<file name>`

    b. I recommend copying the file to your container's home directory instead of root directory. For example, I ran this command:

`docker cp ~/Downloads/InfoSecDemo.sql <container ID>:/root/InfoSecDemo.sql`

        i. Your source path will likely be different than mine, especially if your host is not Linux.

    c. You can close this terminal instance when your copy finishes.
2. In the instance of the terminal that is interfacing with your container, run the following command to import the database into its MySQL instance:

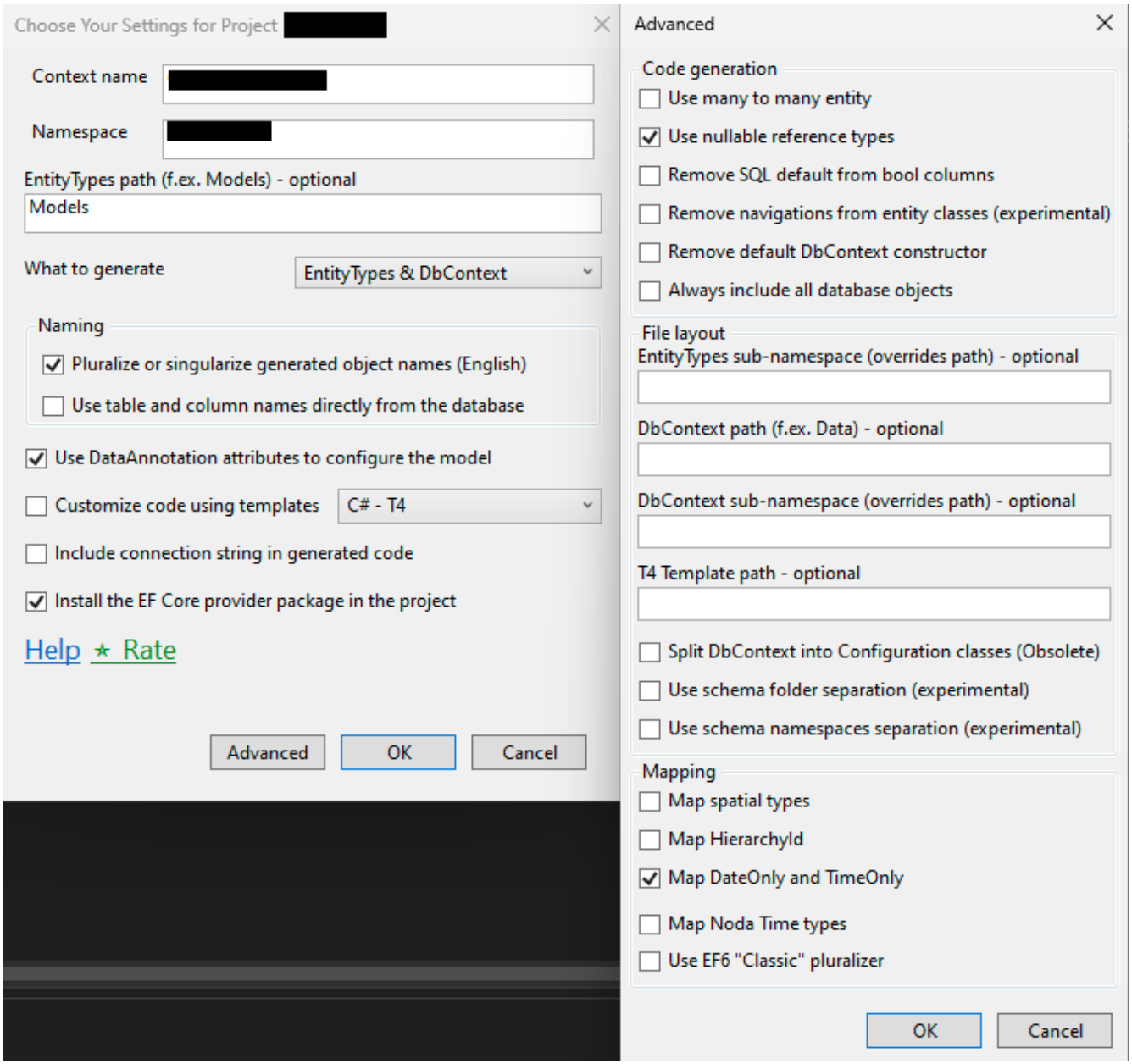
`mysql -u <MySQL username> -p < <path to InfoSecDemo.sql>/InfoSecDemo.sql`

3. Clone and open the Blazor project in the assignment GitHub repository.
4. Install the following NuGet packages to the project:
    a. Microsoft.EntityFrameworkCore
    b. Microsoft.EntityFrameworkCore.Abstractions
    c. MySql.EntityFrameworkCore
5. Add the following to the top of appsettings.json:

```
"ConnectionStrings": {
  "DefaultConnection": "Server=127.0.0.1;Port=3306;Database=InfoSecDemo;
User Id=<your MySQL user>;Password=IT231Pwd!;"
},
```

    a. If you are using a shared connection (not port forwarding), the IP address should be the IP address your VM is using.

6. Scaffold a database context.
    a. (Visual Studio only) Install/update the EF Core Power Tools extension.
        i. After the extension is installed, right-click on the project file, hover over the “EF Core Power Tools”, and select “Reverse Engineer”.
        ii. Add a connection by clicking on the “Add…” button and then “Add Custom Connection”.
            1. Paste your connection string in the “Connection String” box and provide a name to your connection.
        iii. Select all database tables.
        iv. Use these settings:



b. (Rider only) Right-click on the project file, select “Entity Framework Core” then “Scaffold DbContext”.
    i. If you get a warning about a package missing, install it.
    ii. Paste the connection string into the “Connection” text box.
    iii. Set the “Provider” to “MySQL”.
    iv. Check “Use attributes to generate the model”.
    v. Click “OK”.

7. Add the following code to the Program class’ Main() method, directly above the line that creates a `WebApplication` instance. This will register the service and utilize the connection string made earlier.

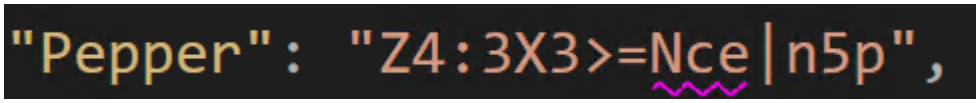
```
builder.Services.AddDbContext<AppDbContext>(options =>
{
      options.UseMySQL(builder.Configuration
            .GetConnectionString("DefaultConnection")!);
#if DEBUG
      options.EnableSensitiveDataLogging();
      options.EnableDetailedErrors();
#endif
});
```

a. Replace the type parameter `AppDbContext` with whatever database context your database scaffolding tool made for you.
    i. Whenever AppDbContext is shown in this handout, replace it with whatever database context your database scaffolding tool made for you.

8. Add a random pepper to your program.
    a. Navigate to the following website: https://randomkeygen.com/
    b. Add the following underneath your connection strings in your appsettings.json file:

```
"Pepper": "<any key>",
```

i. Replace <any key> with a strong password from the website linked above. If it breaks your JSON, choose a different strong password.
ii. Example:



c. In your Program class, add the following property:

```
public static String Pepper { get; private set; }
```

d. Before the call to `app.Run()` in the Program class, add the following line:

```
Pepper = app.Configuration["Pepper"]!;
```

Next, we'll write our code to hash passwords, however, it's generally not recommended to create your own hashing algorithm for password hashing. Designing a secure hashing algorithm requires a deep understanding of cryptographic principles and security best practices. Most developers lack this expertise, making homemade algorithms prone to vulnerabilities that could be exploited by attackers.

Instead of making our own hashing algorithm, we'll leverage an existing one called bcrypt. Bcrypt is a key derivation function specifically designed for password hashing. It's based on the Blowfish cipher and is notable for its adaptive nature, which allows it to adjust its computational cost over time, making it resistant to brute-force attacks as hardware becomes faster.

Bcrypt incorporates the generation and management of a salt directly into its hashing process, which eliminates the need for developers to handle salts separately or maintain a separate salt column in our database.

Bcrypt is widely adopted and supported across various programming languages and frameworks.

9. Install the following NuGet package to the project:
    a. BCrypt.Net-Next
        i. NOT BCrypt.Net. This is an old version.
10. Create a new class called HashUtils under the project.
    a. Make the class static so it cannot be instantiated.
11. Write a method to hash passwords. It should look like the method below:

```
public static String HashPassword(String password)
{
    String pepperedPassword = password + Program.Pepper;
    return BCrypt.Net.BCrypt.EnhancedHashPassword(pepperedPassword);
}
```

12. Complete the registration functionality. You can find the file to modify at Components/Pages/Login/Register.razor.cs. Read the following subpoints before starting.
    a. There is no need to modify the Register.razor page. When the registration form is submitted, the `PerformRegister()` method will automatically be called. The Email and Password properties should have values when the form is submitted.
    b. Utilize the `HashPassword()` method made earlier.
    c. Ensure that no duplicate emails can be inserted into the database.
    d. Add the following code to your Register class:

```
[Inject]
private AppDbContext DbContext { get; set; }
```

        i. You will need the `Microsoft.AspNetCore.Components` namespace.

e. Call the `RegistrationSuccessful()` method if there is a successful registration.

f. Be sure to check your tables to make sure rows are being inserted correctly.

   i. If they are not, ensure that after you add to the Users table, you then save database changes.

g. Ensure that duplicate passwords have different hash results.

Now that users can register, we'll implement the login functionality. To achieve this, we need to be able to accept a password from a user and validate it against hashed passwords. Let's add the code to validate passwords first.

13. In the HashUtils class, add the following method to verify passwords:

```
public static bool VerifyPassword(String password, String storedHash)
{
    String pepperedPassword = password + Program.Pepper;
    return BCrypt.Net.BCrypt.Verify(pepperedPassword, storedHash, true);
}
```

    a. The last argument being 'true' is important. The hashing algorithm we've employed uses enhanced entropy, therefore, when we are verifying passwords, we must denote that we are using enhanced entropy.

14. Complete the login functionality. You can find the file to modify at Components/Pages/Login/Index.razor.cs. Read the following subpoints before starting.

    a. Just as the Register page, there is no need to modify the Index.razor page. When the login form is submitted, the `PerformLogin()` method will automatically be called. The Email and Password properties should have values when the form is submitted.

    b. Add the following code to your Index class:

```
[Inject]
private AppDbContext DbContext { get; set; }
```

       i. You will need the `Microsoft.AspNetCore.Components` namespace.

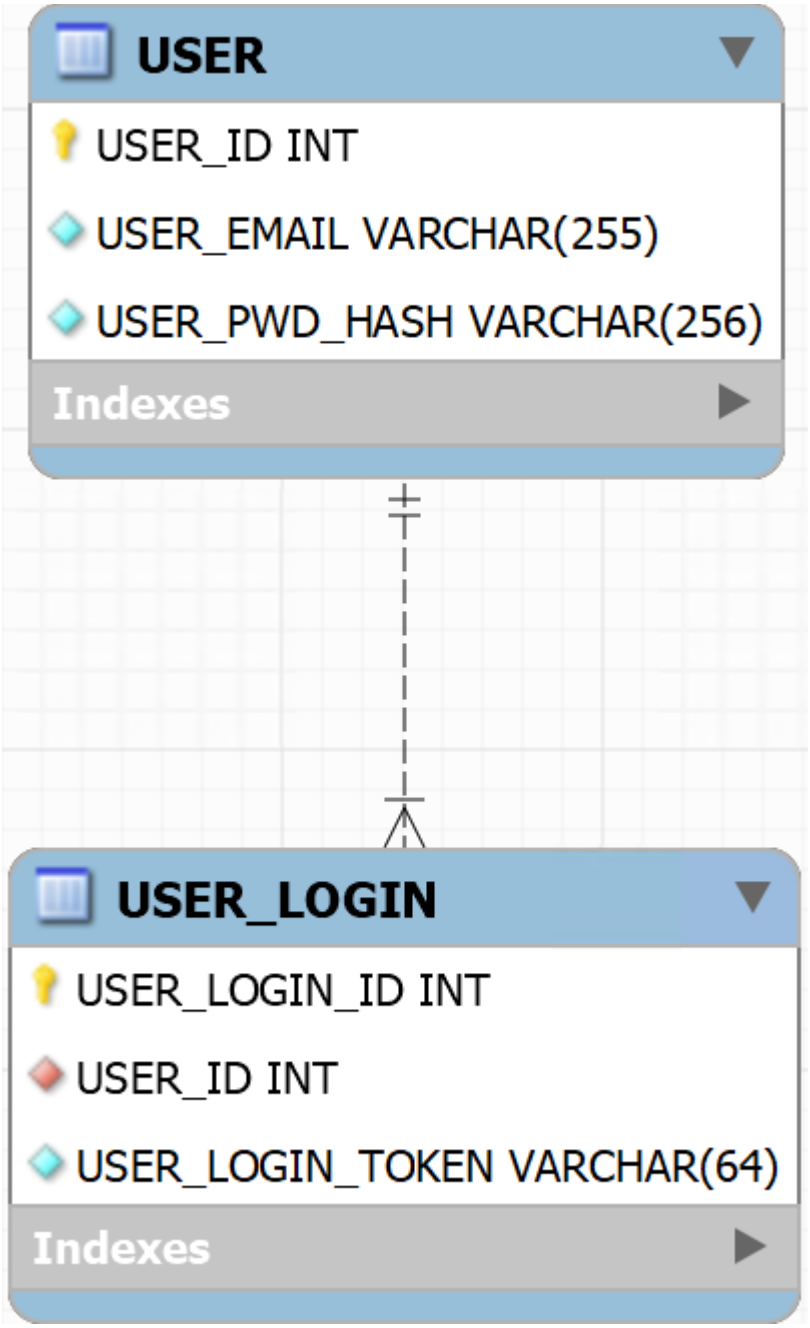
    c. Call the `LoginSuccessful()` method if there is a successful login.

Typically, additional actions should occur once a user is logged in, such as generating and storing a unique token on the client's device. This token would serve to authenticate the user in subsequent requests.

Submission

The submission for this lab is the commit ID you want graded. Submit the commit ID on Canvas.

InfoSecDemo Information



- No columns allow null.
- There is no explicit column for the salt. This will be explained in the lab.
- USER_ID in the USER_LOGIN table is a FK to USER.
- USER_ID (in USER and USER_LOGIN) is an unsigned integer.
- USER_LOGIN_ID is an unsigned integer.